Core Java

Akshita Chanchlani

Agenda

- Exception Handling

Operating System Resources

- Following are the operating system resources that we can use it in the program:

  1. Memory(RAM)

  2. File

  3. Thread

  4. Socket

  5. Connection

  6. IO Devices etc.

- Since OS resources are limited, we should handle it carefully. In other words, we should avoid their leakage.

Resource Type and resource in Java

- AutoCloseable is interface declared in java.lang package.

- Methods:

    1. void close() throws Exception

    2. This method is invoked automatically on objects managed by the try-with-resources statement.

- java.io.Closeable is sub interface of java.lang.AutoCloseable interface.

- Methods:

    1. void close() throws IOException

    2. This method is invoked automatically on objects managed by the try-with-resources statement.

Resource Type and resource in Java

```java
//Class Test => Resource Type
class Test implements AutoCloseable{
    private Scanner sc;
    public Test() {
        this.sc = new Scanner(System.in);
    }
    //TODO
    @Override
    public void close() throws Exception {
        this.sc.close();
    }
}
public class Program {
    public static void main(String[] args) {
        Test t = null;
        t = new Test( );     //Resource
    }
}
```

Resource Type and resource in Java

- In the context of exception handling, any class which implements java.lang.AutoCloseable or its sub interface(e.g. java.io.Closeable) is called resource type and its instance is called as resource.

- We can use instance of only resource type inside try-with-resource.

- java.util.Scanner class implements java.io.Closeable interface. Hence Scanner class is called as resource type.

Exception Handling
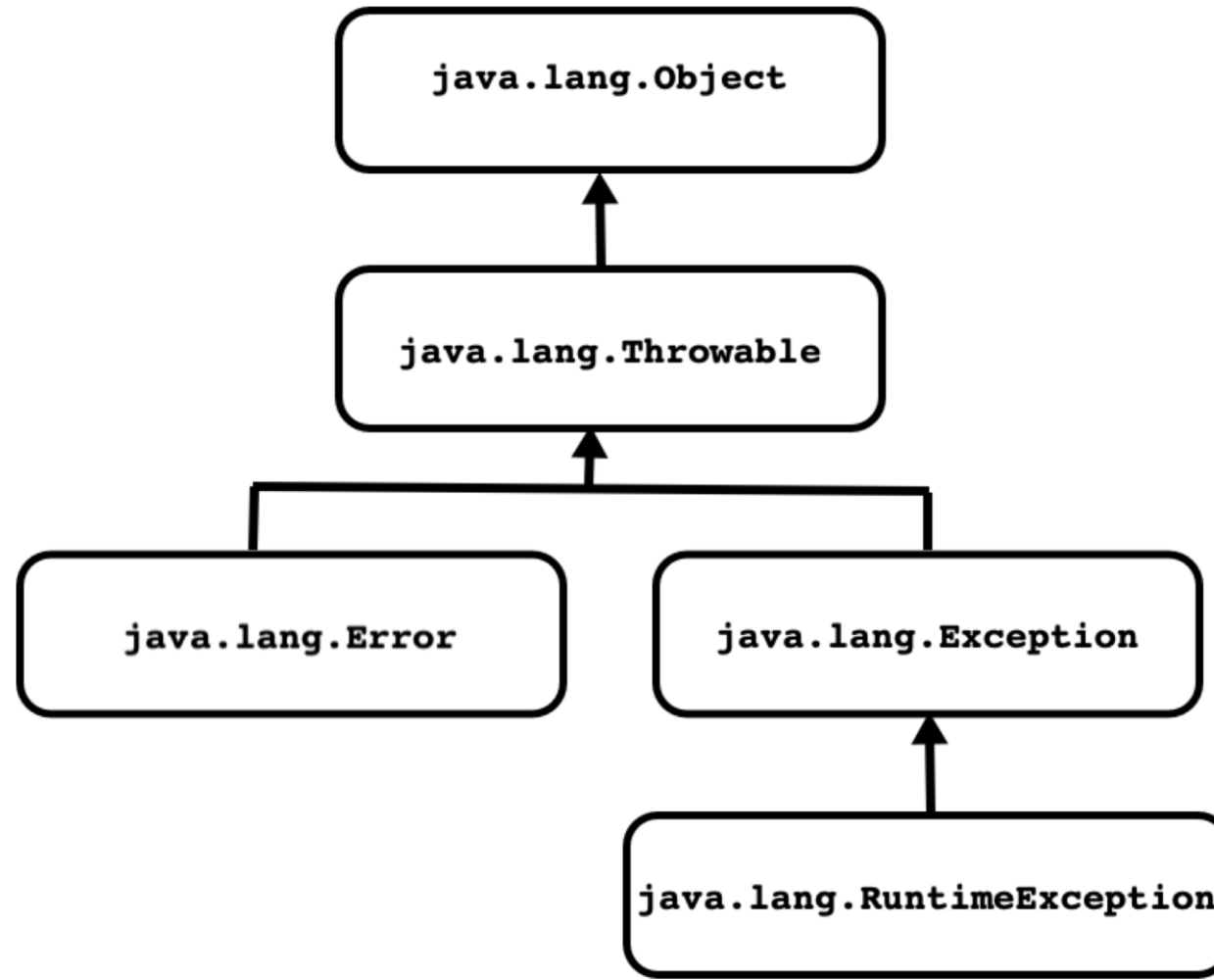
- **Why we should handle exception**

  1. To handle all runtime errors at single place. It helps developer to reduces maintenance.

  2. To avoid resource leakage/ to manage OS resources carefully.

- **How can we handle exception in Java?**

  1. try

  2. catch

  3. throw

  4. throws

  5. finally

Exception Handling

Throwable Class

- It is a class declared in java.lang package.

- The Throwable class is the super class of all errors and exceptions in the Java language.

- Only instances that are instances of Throwable class (or one of its subclasses) are thrown by the Java Virtual Machine or can be thrown by the Java throw statement.

```java
throw 0;      //Not OK

int x = 0;
throw x;      //Not OK

class Test{
}
throw new Test( );   //Not OK

class MyExcetion extends Throwable{
}
throw new MyException();      //OK
```

Throwable Class

- Constructors of Throwable class:

```
1. public Throwable()
       Throwable t1 = new Throwable( );

2. public Throwable(String message)
       Throwable t1 = new Throwable( "exception message" );

3. public Throwable(Throwable cause)
       Throwable cause = new Throwable( );
       Throwable t1 = new Throwable( cause );

4. public Throwable(String message, Throwable cause)
       Throwable cause = new Throwable( );
       Throwable t1 = new Throwable( "exception message", cause );
```

Throwable Class

- **Methods of Throwable class:**
  1. public Throwable initCause(Throwable cause)
  2. public Throwable getCause()
  3. public String getMessage()
  4. public void printStackTrace()
  5. public void printStackTrace(PrintStream s)
  6. public void printStackTrace(PrintWriter s)

Error

- java.lang.Error is a sub class of Throwable class.

- It gets generated due to environmental condition/Runtime environment(For Example, problem in RAM/JVM, Crashing HDD etc.).

- We can not recover from error hence we should not try to catch error. But can write try-catch block to handle error.

- Example:

    1. VirtualMachineError

    2. OutOfMemoryError

    3. InternalError

    4. StackOverflowError

Exception

- `java.lang.Exception is a sub class of Throwable class.`

- `It gets generated due to application.`

- `We can recover from exception hence it is recommended to write try-catch block to handle exception in Java.`

- `Example:`
    1. `NumberFormatException`
    2. `NullPointerException`
    3. `NegativeArraySizeException`
    4. `ArrayIndexOutOfBoundsException`
    5. `ArrayStoreException`
    6. `IllegalArgumentException`
    7. `ClassCastException`

Types Of Exception

- **Unchecked Exception**
    - o java.lang.RuntimeException and all its sub classes are considered as unchecked exception.
    - o **It is not mandatory to handle unchecked exception.**
    - o Example:
        1. NullPointerException
        2. ClassCastException
        3. ArrayIndexOutOfBoundsException
    - o During the execution of arithmetic operation, if any exceptional situation occurs then JVM throws ArithmeticException.
- **Checked Exception**
    - o java.lang.Exception and all its sub classes except java.lang.RuntimeException are considered as checked exception.
    - o **It is mandatory to handle checked exception.**
    - o Example:
        1. java.lang.CloneNotSupportedException
        2. java.lang.InterruptedException

Exception Handling

- **try**

    o It is a keyword in Java.

    o If we want to keep watch on statements for the exception then we should put all such statements inside try block/handler.

    o try block must have at least one:

        1. catch block or

        2. finally block or

        3. Resource

    o We can not define try block after catch or finally block.

Exception Handling

- **Catch**

  - It is a keyword in Java.

  - If we want to handle exception then we should use catch block/handler

  - Only Throwable class or one of its subclasses can be the argument type in a catch clause.

  - Catch block can handle exception thrown from try block only.

  - For single try block we can define multiple catch block.

  - Multi-catch block allows us to handle multiple specific exception inside single catch block.

```java
try {
    //TODO
}catch (ArithmeticException | InputMismatchException e) {
    e.printStackTrace( );
}
```

Exception Handling

Let us consider hierarchy of ArithmeticException class:
- java.lang.Exception
    - java.lang.RuntimeException
        - java.lang.ArithmeticException


```
ArithmeticException e1 = new ArithmeticException( );      //OK
RuntimeException e2 = new ArithmeticException( );    //OK : Upcasting
Exception e3 = new ArithmeticException( );    //OK : Upcasting
```

Let us consider hierarchy of InterruptedException class:
- java.lang.Exception
    - java.lang.InterruptedException


```
InterruptedException e1 = new InterruptedException( );
Exception e2 =  new InterruptedException( ); //OK : Upcasting
```

Exception Handling

- A catch block, which can handle all type of exception is called generic catch block.

- Exception class reference variable can contain reference of instance of any checked as well as unchecked exception. Hence to write generic catch block, we should use java.lang.Exception class.

```
try{


}catch( Exception ex ){ //Generic catch block

    ex.printStackTrace( );

}
```

Exception Handling

- In case of hierarchy, It is necessary to handle all sub type of exception first.

```java
try {
    //TODO
}catch (ArithmeticException e) {
    e.printStackTrace();
}catch (RuntimeException e) {
    e.printStackTrace();
}catch (Exception e) {
    e.printStackTrace();
}
```

Exception Handling

- **`throw`**

    o It is a keyword in Java.

    o If we want to generate new exception then we should use throw keyword.

    o Only objects that are instances of Throwable class (or one of its subclasses) are thrown by the Java Virtual Machine or can be thrown by the Java throw statement.

    o throw statement is a jump statement.

Exception Handling

- **finally**

  o It is a keyword in Java.

  o If we want to release local resources then we should use finally block.

  o We can not define finally block before try and catch block.

  o Try block may have only one finally block.

  o JVM always execute finally block.

  o If we call System.exit(0) inside try block and catch block then JVM do not execute finally block.

Exception Handling

- **throws**

  - o It is a keyword in Java.

  - o If we want to redirect/delegate exception from one method to another then we should use throws clause.

  - o Consider declaration of following methods:

    1. public static int parseInt(String s) throws NumberFormatException

    2. public static void sleep(long millis) throws InterruptedException

Exception Handling

- **`try-with-resources`**

  o The try-with-resources statement is a try statement that declares one or more resources.

  o A   is an object that must be closed after the program is finished with it.

  o The try-with-resources statement ensures that each resource is closed at the end of the statement.

  o Any object that implements java.lang.AutoCloseable, which includes all objects which implement java.io.Closeable, can be used as a resource.

```java
public static String readFirstLineFromFile(String path) throws IOException {
    try (BufferedReader br = new BufferedReader(new FileReader(path))) {
        return br.readLine();
    }
}
```

Custom Exception

- JVM can not understand, exceptional situations/conditions of business logic. If we want to handle such exceptional conditions then we should use custom exceptions.

Custom unchecked exception

```java
class StackOverflowException extends RuntimeException{
    //TODO
}
```

Custom checked exception

```java
class StackOverflowException extends Exception{
    //TODO
}
```

Thank you.
akshita.Chanchlani@sunbeaminfo.com